A Chaining Algorithm for Mapping cDNA Sequences to Multiple Genomic Sequences

Mohamed I. Abouelhoda
Faculty of Engineering and Computer Science, Ulm University
D-89069 Ulm, Germany.
Email: mohamed.ibrahim@uni-ulm.de

***Abstract*— Given a set of matches between a cDNA sequence and *multiple* genomic sequences (k-dimensional fragments), we present the first subquadratic chaining algorithm for computing an optimal chain of colinear matches, while tolerating overlaps between the fragments. The fragments of the resulting chain serve as anchors for locating the cDNA in the given genomic sequences and for producing a multiple alignment. Moreover, the resulting chains identify the common genes among the given sequences and in the same time identify the syntenic regions (regions of conserved gene-order). Our algorithm is a new addition to the family of chaining algorithms permitting overlaps, because the previous algorithms are limited to fragments between a cDNA sequence and a *single* genomic sequence.**

I. Introduction

A fundamental task of every genome annotation project is to locate each gene in the genome and to determine its structure. This knowledge serves as a basis for elucidating the gene function, detecting the regulatory elements, and studying the genome organization and evolution. One of the most successful methods for accomplishing this task is the mapping of cDNA sequences to the genomes they are transcribed from. A cDNA sequence is a complementary sequence to an mRNA. (The introns are spliced out.) Consequently, a perfect alignment of a cDNA sequence to the related genomic sequence locates the corresponding gene and directly reveals its exon-intron structure; see Figure 1. The increasing number of full cDNA sequencing projects reflects the increasing popularity of this method.

In [11], Shibuya and Kurochkin have presented an efficient algorithm to map a cDNA sequence to a single large genomic sequence. Their algorithm is based on a strategy that is composed of three phases: First, fragments of the type (rare) maximal exact matches are computed using the suffix tree in linear time and space. Second, a geometry based chaining algorithm finds a highest scoring chain of colinear fragments in $O(m \log m)$ time and $O(m)$ space, where m is the number of the fragments. The novel part of this chaining algorithm is that overlaps between the fragments of the chain are tolerated to improve the chain coverage, while the worst case complexity is the same as the algorithms tolerating no overlap. (Algorithms permitting *no* overlaps have been presented in [1, 6, 9, 12].)

Although the algorithm is relatively complicated due to the combination of range maximum queries (RMQs) and the candidate list paradigm, it is an important improvement over the graph based solution that takes $O(m^2)$ time [11]. Finally,

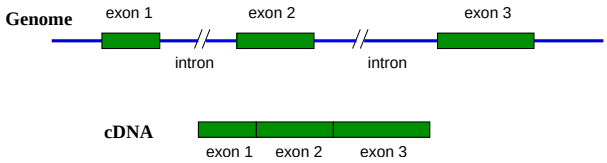

Fig. 1. cDNA mapped to a genomic sequence.

an alignment is produced by applying a traditional alignment algorithm (considering the exon-intron boundaries) on the regions between the fragments of the chain. The rationale behind permitting overlaps is that (1) overlapping fragments were found to be common in cDNA mapping, and usually occur at the exon-intron boundaries [11], (2) the chain coverage will improve, which is crucial for both increasing the sensitivity/specificity of the procedure and for speeding-up the mapping task. Regarding the sensitivity/specificity, permitting no overlapping in the chain will cause many fragments to be discarded from the respective chains, which probably leads to filtering out many of these chains, despite being significant. This is because the scores of these chains drop below the user-defined threshold. If the user attempts to overcome this drawback and decreases the threshold, many insignificant chains will not be filtered out and the specificity will decrease. As for the running time, reduced chain coverage results in spending much time in the third phase in which an alignment on the character level is computed to finish the mapping task.

With the ever increasing number of sequenced genomes of very closely related species or of different strains of the same species, it is natural to extend the algorithm of Shibuya and Kurochkin to map a cDNA sequence to multiple genomes. This provides a method for identifying the common genes among the genomes and for identifying the syntenic regions (regions of conserved gene-order).

Generating fragments from multiple genomic sequences can be easily achieved in linear time and space, using the suffix tree [8] or the enhanced suffix array [2]. Chaining fragments from k sequences (one of them being cDNA) while permitting overlaps can be generally done in $O(m^2)$ time by a straightforward modification of the graph-based approach of [8]. Unfortunately, the quadratic running time of this approach is a serious drawback for a large number of fragments. To overcome this obstacle, one has to use a

geometry-based solution to chain the fragments from k sequences in subquadratic time. Unfortunately, the extension of the algorithm of Shibuya and Kurochkin is very complicated, if not infeasible, due to the difficulty of analyzing the overlaps, according to the suggested objective function, and due to the difficulty of combining the corresponding RMQs and candidate lists; Shibuya and Kurochkin noticed also these complications [11]. Nevertheless, we show in this paper that an efficient solution exists, if an objective function specific to the cDNA mapping problem is used. We present a chaining algorithm that runs in subquadratic time, which is a significant improvement over the naive graph-based algorithm. Our algorithm is easy to implement, because it uses solely RMQs without candidate lists.

In the following section, the basic concepts and definitions used in this paper are presented. Section III introduces the chaining problem and discuss a graph based solution. In Section IV, we present our geometry based solution. Section V contains conclusions and future work.

II. Basic concepts and definitions

Let $S[1..n]$ denote a string of length n. $S[i..j]$ denotes the substring of $S[1..n]$ starting at position i and ending at position j in S, and $S[i]$ denotes the i^{th} character of S. The string S in our application represents a DNA sequence (cDNA or a genomic sequence). A *multiple exact match* between k sequences $S_1, .., S_k$ is a $(k+1)-$tuple $(l, p_1, .., p_k)$ such that $S_1[p_1..p_1+l-1] = \ldots = S_k[p_k..p_k+l-1]$, i.e., the l-character-long substrings of $S_1, .., S_k$ starting at positions $p_1, .., p_k$, respectively, are identical. A multiple exact match is *left maximal* if $S_i[p_i-1] \neq S_j[p_j-1]$ for any $1 \leq i \neq j \leq k$, and *right maximal* if $S_i[p_i+l] \neq S_j[p_j+l]$ for any $1 \leq i \neq j \leq k$, i.e., it cannot be extended to the left and to the right simultaneously in all the sequences. A *maximal multiple exact match* (*multi-MEM*) is a left and right maximal multiple exact match.

A *multi-MEM* $(l, p_1, .., p_k)$ can be represented by a hyper-rectangle in $\mathbb{R}^k$ with the two extreme corner points $(p_1, .., p_k)$ and $(p_1+l-1, .., p_k+l-1)$. Such a hyper-rectangle is also called a *fragment*. Figure 3 shows an example of a two dimensional fragment, i.e., for $k=2$. In the following, we will identify a *multi-MEM* $(l, p_1, .., p_k)$ with its fragment f in $\mathbb{R}^k$ and denote the two extreme corner points by $beg(f) = (beg(f).x_1, .., beg(f).x_k) = (p_1, .., p_k)$ and $end(f) = (end(f).x_1, .., end(f).x_k) = (p_1+l-1, .., p_k+l-1)$. Furthermore, we define $f.length = l$ to denote the length of the *multi-MEM* corresponding to f.

For ease of presentation, we consider the point $\mathsf{0} = (0, \ldots, 0)$ (the origin) and the terminus $\mathsf{t} = (|S_1|-1, \ldots, |S_k|-1)$ as fragments with weight 0. For these fragments, we define $beg(\mathsf{0}) = \perp$, $end(\mathsf{0}) = \mathsf{0}$, $beg(\mathsf{t}) = \mathsf{t}$, and $end(\mathsf{t}) = \perp$, where $\perp$ stands for an undefined value.

multi-MEMs can be readily computed by means of an index data structure, such as the suffix tree or the enhanced suffix array, using any of the algorithms in [2, 4, 5, 7, 8, 10]. All these algorithms take linear time and space. However, they differ in the kind of matches (some compute *multi-MEMs* from two sequences only and some compute *multi-MEMs* from multiple sequences.) and in the programming details through which the matches are generated.

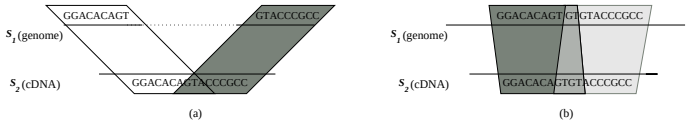

Fig. 2. Overlapping *multi-MEMs* of two sequences.

III. Chaining fragments with overlaps

Definition 3.1: Let $(l_1, p_1, .., p_k)$ and $(l_2, q_1, .., q_k)$ be two *multi-MEMs* with $p_i < q_i$, $1 \leq i \leq k$. We say that $(l_1, p_1, .., p_k)$ *overlaps* with $(l_2, q_1, .., q_k)$ in S_i if and only if $q_i \leq p_i + l_1 - 1 < q_i + l_2 - 1$, for any $1 \leq i \leq k$.

For $k=2$, Figure 2 (a) shows an example of two *multi-MEMs* overlapping in S_2 but not in S_1, while Figure 2 (b) shows two *multi-MEMs* overlapping in both S_1 and S_2.

Definition 3.2: The relation $\ll$ on the set of fragments is defined as follows. $f' \ll f$ if and only if the following two conditions hold: $beg(f').x_i < beg(f).x_i$ and $end(f').x_i < end(f).x_i$, for all $1 \leq i \leq k$. If $f' \ll f$, then we say that f' *precedes* f. The fragments f' and f are *colinear* if either f' precedes f or f precedes f'.

Thus, two fragments are colinear if they appear in the same order in both sequences. Note that if we further have $end(f').x_i < beg(f).x_i$, for all $1 \leq i \leq k$, then f' and f are colinear and non-overlapping.

For two sequences, Shibuya and Kurochkin [11] defined the *overlap length* of two fragments to be the *maximum* of the amount of overlap in S_1 and in S_2. In their paper, the score $score(C)$ of a chain C of colinear fragments is the sum of the lengths of the fragments in C minus their overlap lengths. (Note that there is no gap cost between the matches because the large gaps correspond to introns.) As a consequence of their definition of the overlap length in the computation of a highest-scoring chain C_f ending with f, one has to consider four different cases corresponding to the regions A, $B \cup C_1$, C_2, and D shown in Figure 3 (a). Each of these cases yields a candidate fragment f_i, found either by RMQs over the corresponding region or by maintaining candidate lists, and a highest-scoring chain C_{f_i} of colinear fragments ending with f_i, $1 \leq i \leq 4$. Then $score(C_f)$ is computed by $score(C_f) = \max_i\{score(C_{f_i}) + f.length - overlap\ length(f, f_i)\}$; see [11] for details. This algorithm takes $O(m \log m)$ time and $O(m)$ space, where m is the number of fragments. Unfortunately, a direct extension of this algorithm to k sequences, one of them being cDNA, is very complicated because (1) the suggested penalty for overlaps results in complicated non-orthogonal space division, and (2) the combination of the RMQs and the candidate list paradigm becomes difficult to realize for $k > 2$. In the following, we overcome these problems by (1) presenting a measure

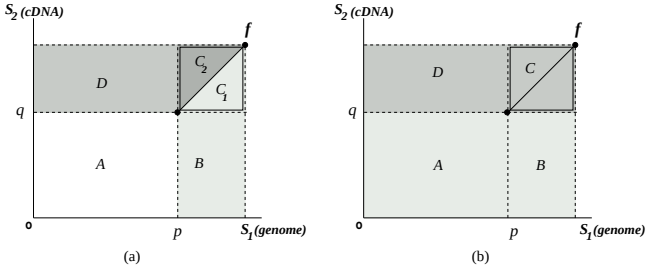

Fig. 3. The fragment f is represented by a 2D rectangle whose extreme corner points $beg(f)$ and $end(f)$ are drawn as black circles. (a) The four cases that have to be considered in Shibuya and Kurochkin's approach. (b) The two cases $A \cup B$ and $C \cup D$ that have to be considered in our approach.

of overlaps that considers the fact that one of the sequences is cDNA, (2) formulating the objective function as the one that maximizes the coverage of the cDNA w.r.t. the genomic sequences, and (3) using solely RMQs.

Definition 3.3: For any two fragments f and f' in k dimensional space, where the k^{th} axis corresponds to the cDNA sequence, the amount of overlap in the cDNA sequence is

$$overlap_k(f', f) = \begin{cases} end(f').x_k - beg(f).x_k + 1, \\ \quad \text{if } beg(f).x_k \leq end(f').x_k \leq end(f).x_k \\ 0, \text{ otherwise} \end{cases}$$

Based on this measure of overlap, the cDNA chaining problem can be formulated as follows.

Definition 3.4: Given a set of m fragments, find a chain C of colinear fragments $f_1, f_2, .., f_t$ (i.e., $f_1 \ll f_2 \ll .. \ll f_t$) such that $score(C) = \sum_{i=1}^{t} f_i.length - \sum_{i=1}^{t-1} overlap_k(f_i, f_{i+1})$ is maximal.

This objective function maximizes the amount of cDNA sequence mapped to the genomic sequence. It is easy to see that a perfect mapping has a score that equals the cDNA length (percentage coverage being 100%). This function does not conflict with aligning the sequences on the character level, because overlaps can always be resolved by modifying the involved *multi-MEMs* boundaries and/or performing character indels of zero cost in the genomic sequences. Computationally, this objective function has the advantage, w.r.t. the space devision around a fragment f, that only two regions (defined soon) have to be considered, independently of k. As we will show later, this means that the chaining problem for fragments from k sequences, one of them being cDNA, can be solved by only two RMQs in these regions.

A straightforward solution to the cDNA chaining problem is to construct a weighted directed acyclic graph $G(V, E)$, where the set of vertices V is the set of fragments (including 0 and t), and the set of edges E is characterized as follows. For any two nodes $v' = f'$ and $v = f$, there is an edge $e(v' \rightarrow v) \in E$ with weight of $f.length - overlap(f', f)$, only if $f' \ll f$. An optimal chain of fragments corresponds to a path with maximum score from vertex 0 to vertex t in the graph. Because the graph is acyclic, such a path can be computed as follows. Let $f.score$ denote the maximum score of all chains ending with the fragment f. When we speak about the score of the points $beg(f)$ and $end(f)$ we implicitly mean the score of f. Clearly, $f.score$ can be computed by the recurrence

$$f.score = f.length + \max\{f'.score - overlap_k(f', f) | f' \ll f\}$$

A dynamic programming algorithm based on this recurrence takes $O(|V| + |E|)$ time provided that computing overlaps takes constant time. Because $|V| + |E| \in O(m^2)$, computing an optimal chain takes quadratic time. This quadratic running time of this graph based approach is a drawback for a large number of fragments. In the following section, we present a geometry based solution that runs in subquadratic time.

IV. Geometric based Solution

Algorithm 4.1 is a geometric solution to this recurrence. The algorithm assumes that the k^{th} axis corresponds to the cDNA. It is based on a sweep-line procedure w.r.t. the first genomic sequence and it uses RMQs to find the fragment that maximizes the score. There are two data structures D_1 and D_2 to answer $(2k-2)$-dimensional RMQs with activation. Each fragment f is represented in D_1 and D_2 by the $(2k-2)$-dimensional point $(end(f).x_1, .., end(f).x_k, beg(f).x_2, .., beg(f).x_{k-1})$. For example, for $k = 3$ f is represented by the 4-dimensional point $(end(f).x_1, end(f).x_2, end(f).x_3, beg(f).x_2)$. In this algorithm, the function $RMQ_{D_j}([p_1..q_1], .., [p_k..q_k], [p'_2..q'_2], .., [p'_{k-1}..q'_{k-1}])$, $j \in \{1, 2\}$, is a range maximum query that retrieves the fragment of maximum score in the set of active fragments/points stored in the data structure D_j. By the construction of D_1 and D_2, this fragment has its end point within the hyper-rectangular region defined by the intervals $[p_1..q_1], .., [p_k..q_k]$ in $S_1, .., S_k$, respectively, and its start point within the hyper-rectangle defined by the intervals $[p'_2..q'_2], .., [p'_{k-1}..q'_{k-1}]$ in $S_2, .., S_{k-1}$, respectively. As we will see soon, the restriction added on the fragment start points guarantees that the fragment to be connected to f precedes f. As we will discuss later D_1 and D_2 can be efficiently implemented either as range trees or *kd*-trees.

Algorithm 4.1:

Sort all start points of the m fragments in ascending order w.r.t. their x_1 coordinate and store them in the array points.
For each fragment f, create the point $(end(f).x_1, .., end(f).x_k, beg(f).x_2, .., beg(f).x_{k-1})$ and store it as inactive in the data structures D_1 and D_2.
for $1 \leq i \leq m$
  determine the fragment f with $beg(f).x_1 = \text{points}[i]$
  $(b.x_1, .., b.x_k) := (beg(f).x_1, .., beg(f).x_k)$
  $(e.x_1, .., e.x_k) := (end(f).x_1, .., end(f).x_k)$
  $q_1 := \text{RMQ}_{D_1}([0..e.x_1 - 1], .., [0..e.x_{k-1} - 1], [0..b.x_k - 1], [0..b.x_2 - 1], .., [0..b.x_{k-1} - 1])$
  $q_2 := \text{RMQ}_{D_2}([0..e.x_1 - 1], .., [0..e.x_{k-1} - 1], [b.x_k..e.x_k - 1], [0..b.x_2 - 1], .., [0..b.x_{k-1} - 1])$
  determine the fragments f_1 and f_2 corresponding to q_1 and q_2, respectively
  if $f_1 = \perp$ **then** $score_1 = 0$
    /$\star$ $\perp$ means no fragment found in the region $\star$/
  else $score_1 = f_1.score$
  if $f_2 = \perp$ **then** $score_2 = 0$
    /$\star$ $\perp$ means no fragment found in the region $\star$/
  else
    if $beg(f_2).x_k < beg(f).x_k$ **then**
      $score_2 = f_2.score - (end(f_2).x_k - beg(f).x_k)$
    else $f_2 = \perp$, and $score_2 = 0$

$f.score = f.length + \max\{score_1, score_2\}$
if $score_1 \geq score_2 > 0$ **then** connect f_1 to f
else if $score_2 > 0$ **then** connect f_2 to f
activate $(e.x_1, .., e.x_k, b.x_2, .., b.x_{k-1})$ in D_1 with score $f.score$
activate $(e.x_1, .., e.x_k, b.x_2, .., b.x_{k-1})$ in D_2 with score $(f.score - end(f).x_k)$

Before proving the correctness of this algorithm, we would like to explain it. When $beg(f)$ is scanned, we search for a fragment f' that precedes f and maximizes $f'.score - overlap_k(f', f)$. Geometrically, we search for the fragment f' such that $end(f')$ lies in the region $ABCD(f)$ defined by the hyper-rectangle $([0..end(f).x_1 - 1], .., [0..end(f).x_k - 1])$, provided that $beg(f')$ lies in the region $A(f)$ defined by the hyper-rectangle $([0..beg(f).x_1 - 1], .., [0..beg(f).x_k - 1])$. In Figure 3 (b), the region $ABCD(f)$ is the region $A \cup B \cup C \cup D$, and $A(f)$ is A, where $k = 2$. Note that the notation $ABCD(f)$ (similarly for any subregion of it like $A(f)$) emphasizes the dependence of this region on f. To take overlaps into account, we have to divide the search region $ABCD(f)$ into two subregions. The first subregion is the hyper-rectangle $([0..end(f).x_1 - 1], .., [0..end(f).x_{k-1} - 1], [0..beg(f).x_k - 1])$, denoted by $AB(f)$. Any fragment in this region does not overlap with f in the cDNA sequence. $AB(f)$ is the region $A \cup B$ in Figure 3 (b). In the algorithm, $\mathtt{RMQ}_{D_1}$ finds the fragment $f_1 \ll f$ of highest score in $AB(f)$, as follows. The first k ranges of $\mathtt{RMQ}_{D_1}$ restrict the search space to those fragments with end points in $AB(f)$, and the last $(k-2)$ ranges of $\mathtt{RMQ}_{D_1}$ add the constraint that the start points of these fragments are in $A(f)$. Note that we use only $(k-2)$ ranges for the constraint that the the start points are in $A(f)$, i.e., the ranges $[0..beg(f).x_1]$ and $[0..beg(f).x_k]$ are not explicitly included in $\mathtt{RMQ}_{D_1}$. The reason is that any fragment f' with $beg(f').x_1 > beg(f).x_1$ is not yet activated in D_1, and the k^{th} argument of $\mathtt{RMQ}_{D_1}$, which restricts that $end(f_2).x_k \in [0..beg(f).x_k]$, implies that $beg(f_2).x_k \in [0..beg(f).x_k]$. That is, it is superfluous to include these ranges, and unnecessarily increase the dimensionality of the RMQ. From the fragments satisfying the range constraints, a fragment f_1 of highest score is retrieved. For $k = 3$, e.g., $\mathtt{RMQ}_{D_1}([0..end(f).x_1], [0..end(f).x_2], [0..beg(f).x_3], [0..beg(f).x_2])$ yields the highest scoring fragment f_1. The first three ranges of $\mathtt{RMQ}_{D_1}$ assure that $end(f_1) \in AB(f)$. The fourth range guarantees that $beg(f_1).x_2 \in [0..beg(f).x_2]$, which is enough to assure that $beg(f_1) \in A(f)$.

The second subregion we examine when scanning a fragment f is the hyper-rectangle $([0..end(f).x_1 - 1], .., [0..end(f).x_{k-1} - 1], [beg(f).x_k..end(f).x_k - 1])$, denoted by $CD(f)$. Any fragment ending in this region overlaps with f in the cDNA sequence. $CD(f)$ is the region $C \cup D$ in Figure 3 (b), where $k = 2$. In order to penalize this overlap, we activate each end point in D_2 with $f.score - end(f).x_k$ instead of $f.score$. This guarantees, as we shall see in the correctness proof, that a fragment f_2 will be found such that $f_2.score - overlap_k(f_2, f)$ is maximal in $CD(f)$. The $(2k-2)$-dimensional RMQ over D_2 finds a fragment f_2 of highest score in this region such that $beg(f_2).x_j \in [0..beg(f_2).x_j], 2 \leq j \leq k-1$. The range $[0..beg(f).x_1]$ is not explicitly included in $\mathtt{RMQ}_{D_2}$, because any fragment f' with $beg(f').x_1 > beg(f).x_1$ is not yet activated in D_2. The range $[0..beg(f).x_k]$ is also not included, because if $beg(f_2).x_k \geq beg(f).x_k$, we simply ignore f_2. This does not affect the correctness of the algorithm, as we shall see in the correctness proof, because then there is a fragment in $AB(f)$ whose score is at least as high as that of f_2. Finally in the algorithm, if $f_1.score \geq f_2.score - overlap_k(f, f_2)$, then f_1 is connected to f. Otherwise, f_2 is connected to f.

For a formal correctness proof, we need the following definition and lemmata.

Definition 4.1: The priority of a fragment $\hat{f}$, denoted by $\hat{f}.priority$, is defined as $\hat{f}.priority = \hat{f}.score - end(\hat{f}).x_k$, where the k^{th} axis corresponds to the cDNA.

Lemma 4.1: Let f, f' and f'' be three fragments with $end(f') \in CD(f)$ and $end(f'') \in CD(f)$. We have $f''.priority < f'.priority$ if and only if $f''.score - overlap_k(f'', f) < f'.score - overlap_k(f', f)$.

Proof:

$$f''.priority < f'.priority$$
$$\Leftrightarrow f''.score - end(f'').x_k < f'.score - end(f').x_k$$

by adding $beg(f).x_k$ to both sides, we obtain

$$f''.score - overlap_k(f'', f) < f'.score - overlap_k(f', f)$$

Note that in the lemma $<$ can be replaced with $\leq$. ■

Thus, if f' is a fragment with highest priority in $CD(f)$, then $f'.score - overlap_k(f', f)$ is maximal in $CD(f)$. The priority of a fragment f' is independent of f. Hence, it can be computed in constant time when f' is scanned. This has the advantage that the overlaps between all fragments need not be computed in advance (note that this would yield a quadratic time algorithm).

Lemma 4.2: Let C be a chain composed of the fragments $f_1, .., f_t$. For every index i, $1 \leq i \leq t-1$, we have $f_{i+1}.priority \leq f_i.priority$.

Proof:

$$\begin{aligned} f_{i+1}.priority &= f_{i+1}.score - end(f_{i+1}).x_k \\ &= f_i.score + f_{i+1}.length - overlap_k(f_i, f_{i+1}) - \\ &\quad end(f_{i+1}).x_k \\ &= f_i.score - beg(f_{i+1}).x_k - overlap_k(f_i, f_{i+1}) + 1 \\ &\leq f_i.score - end(f_i).x_k \\ &\leq f_i.priority \end{aligned}$$

Note that if $overlap_k(f_i, f_{i+1}) = 0$, then $f_{i+1}.priority < f_i.priority$. ■

Theorem 4.1: Algorithm 4.1 correctly computes an optimal chain.

Proof: When the sweep-line reaches the start point $beg(f)$ of fragment f, the end points of all fragments that started before $beg(f).x_1$ are already activated in the data structures D_1 and D_2. The end points of the remaining fragments are still inactive. This guarantees that each fragment whose end point lies in $AB(f)$ or $CD(f)$ but whose start point occurs after $beg(f).x_1$ will not be considered in what follows. Because each fragment with end point in region $AB(f)$ does not overlap with f in the cDNA sequence, the $(2k-2)$-dimensional

range query over D_1 retrieves the fragment of highest score in this region and it is guaranteed, as discussed before, that $beg(f_2) \in A(f)$. Analogously, because a fragment f' with end point in region $CD(f)$ overlaps with f in the cDNA sequence, it is activated in D_2 with priority $f'.score - end(f').x_k$. The $(2k-2)$-dimensional range query over D_2 yields the fragment f_2 of highest priority ending in $CD(f)$ as follows: By Lemma 4.1, $f_2.score - overlap_k(f_2, f)$ is maximal in $CD(f)$. Moreover, it is guaranteed, by the sweep-line procedure and by the range query, that $beg(f_2).x_j < beg(f).x_j, 1 \leq j \leq k-1$. However, it is not guaranteed that $beg(f_2).x_k < beg(f).x_k$, which violates that $beg(f_2) \in A(f)$. To handle this, we proceed by case analysis. Suppose first that $beg(f_2).x_k < beg(f).x_k$, i.e., the start point of f_2 lies in $A(f)$. In this case we connect f_2 to f, if $f_2.score - overlap_k(f_2, f) > f_1.score$. Otherwise, we connect f_1 to f, and we are done. Now suppose that $beg(f_2).x_k \geq beg(f).x_k$, i.e., the start point of f_2 lies in $CD(f)$. This implies that $overlap_k(f_2, f) \geq f_2.length$. According to Lemma 4.2, if there is a fragment f' connected to f_2 (i.e., f' is the predecessor of f_2 in a highest-scoring chain ending with f_2), then the end point $end(f')$ of f' must lie in $A(f)$. Hence, $overlap_k(f', f_2) = 0$ and we have

$$\begin{aligned} f'.score &= f'.score - overlap_k(f', f_2) \\ &= f_2.length + f'.score - overlap_k(f', f_2) - f_2.length \\ &= f_2.score - f_2.length \\ &\geq f_2.score - overlap_k(f_2, f) \end{aligned}$$

Recall from Lemma 4.1 that $f_2.score - overlap_k(f_2, f)$ is maximal in $CD(f)$. Now it follows from $f'.score \geq f_2.score - overlap_k(f_2, f)$ in conjunction with $f'.score \leq f_1.score$ that f_2 can be safely ignored. (If f_2 has no predecessor, then f_2 can be also safely ignored by Lemma 4.1 because $f_2.score - overlap_k(f_2, f) = f_2.length - overlap_k(f_2, f) \leq 0$.) ■

The complexity of Algorithm 4.1 depends on the complexity of the RMQs with activation supported by the data structures D_1 and D_2. If D_1 and D_2 are implemented as range trees supported by the technique of fractional cascading and enhanced with priority queues as shown in [1], then the complexity of the algorithm is $O(m \log^{d-1} m \log\log m)$ time and $O(m \log^{d-1} m)$ space, $d = 2k-2$. If the *kd*-tree is used instead of the range tree, then the algorithm takes $O(m^{2-\frac{1}{d}})$ time and $O(m)$ space. Interestingly, the query time of the *kd*-tree can be improved in practice using a set of programming tricks [3]. If the gaps between successive fragments in a chain are constrained to be at most W characters long (e.g., W could be set to the estimated maximum intron length), then RMQ_{D_1} and RMQ_{D_2} have to be limited to the regions $([e.x_1 - W..e.x_1 - 1], .., [e.x_{k-1} - W..e.x_{k-1} - 1], [b.x_k - W..b.x_k - 1], [0..b.x_2 - 1], .., [0..b.x_{k-1} - 1])$, and $([e.x_1 - W..e.x_1 - 1], .., [e.x_{k-1} - W..e.x_{k-1} - 1], [b.x_k..e.x_k - 1], [0..b.x_2 - 1], .., [0..b.x_{k-1} - 1])$, respectively, where b and e are the start and end points of a fragment.

V. CONCLUSIONS

In this paper, we have presented a subquadratic chaining algorithm that permits overlaps between the fragments, which are matches of the type *multi-MEMs*. Our algorithm is easy to implement because it depends on RMQs that can be implemented either using the range tree or the *kd*-tree. In a further research, we have found that the complexity of the algorithm can be improved for two special cases that are of practical interest (script in preparation): The first is the usage of *multi-MUM*s and rare *multi-MEMs* [1]. The usage of *multi-MUM*s or rare *multi-MEMs* will speed-up the whole procedure, because repetitions will be filtered out. Moreover, there will be no need to use any repeat-masking program; a step that takes many hours. The second special case is when the amount of overlap between the fragments is constrained (i.e., allowed overlap length is at most $\ell - 1$ characters, where ℓ is the minimum fragment length). This constraint is practically relevant, because overlaps are usually short. We have found that the time complexity for these two special cases can be reduced by nearly $log^k n$ factor.

REFERENCES

[1] M. I. Abouelhoda and E. Ohlebusch. Chaining algorithms and applications in comparative genomics. *Journal of Discrete Algorithms*, 3(2-4):321–341, 2005.

[2] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Enhanced suffi x arrays and applicationss. In *Handbook of Computational Molecular Biology*. CRC Press, 2006.

[3] J. L. Bently. K-d trees for semidynamic point sets. In *6th Annual ACM Symposium on Computational Geometry*, pages 187–197. ACM, 1990.

[4] A. L. Delcher, A. Phillippy, J. Carlton, and et al. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res.*, 30(11):2478–2483, 2002.

[5] J. S. Deogen, J. Yang, and F. Ma. EMAGEN: An effi cient approach to multiple genome alignment. In *Proc. of APBC*, pages 113–122, 2004.

[6] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic programming. I: linear cost functions; II: convex and concave cost functions . *J. Assoc. Comput. Mach.*, 39:519–567, 1992.

[7] D. Farré, R. Roset, M. Huerta, J. E. Adsuara, and et al. Identifi cation of patterns in biological sequences at the ALGGEN server: PROMO and MALGEN. *Nucleic Acids Res.*, 31(13):3651–3653, 2003.

[8] M. Höhl, S. Kurtz, and E. Ohlebusch. Effi cient multiple genome alignment. In *Proc. of ISMB*, pages 312–320. Bioinformatics, 18(Supplement 1), 2002.

[9] E. W. Myers and W. Miller. Chaining multiple-alignment fragments in sub-quadratic time. *Proc. of SODA*, pages 38–47, 1995.

[10] E. Ohlebusch and S. Kurtz. Effi cient computation of rare multiple maximal exact matches. Manuscript in preparation.

[11] S. Shibuya and I. Kurochkin. Match chaining algorithms for cDNA mapping. In *Proc. of WABI*, volume 2812 of *LNBI*, pages 462–475. Springer Verlag, 2003.

[12] Z. Zhang, B. Raghavachari, R. C. Hardison, and et al. Chaining multiple-alignment blocks. *J. Computional Biology* , 1:51–64, 1994.

[1] A *multi-MEM* $(l, p_1, .., p_k)$ is called *rare* of repeat-value r, if in each S_i the substring $S_i[p_i..p_i + l - 1]$ occurs at most r times, $1 \leq i \leq k$. A *maximal multiple unique match* (*multi-MUM*) is a rare *multi-MEM* such that $r = 1$, i.e., $S_i[p_i..p_i + l - 1]$ occurs exactly once in each S_i.